韩山师范学院实验报告

姓名：林婷婷　　专业：信息管理与信息系统　班级：20181171　　学号：2018117126

科目：　Android 应用开发　　实验日期：2021.6.11

实验题目：

【实验目的】

数据存储全方案，详解持久化技术

【实验环境及方式】

Android Studio

【实验内容及实验结果】

1. 持久化技术

数据持久化就是指将那些内存中的瞬时数据保存到存储设备中，保证即使在手机或计算机关机的情况下，这些数据仍然不会丢失。

保存在内存中的数据是处于瞬时状态的，而保存在存储设备中的数据是处于持久状态的。持久化技术提供了一种机制，可以让数据在瞬时状态和持久状态之间进行转换。

Android 系统中主要提供了 3 种方式用于简单地实现数据持久化功能：文件存储、SharedPreferences 存储以及数据库存储。

2. 文件存储

文件存储是 Android 中最基本的数据存储方式，它不对存储的内容进行任何格式化处理，所有数据都是原封不动地保存到文件当中的，因而它比较适合存储一些简单的文本数据或二进制数据。如果你想使用文件存储的方式来保存一些较为复杂的结构化数据，就需要定义一套自己的格式规范，方便之后将数据从文件中重新解析出来。

① 将数据存储到文件中：

Context 类中提供了一个 openFileOutput()方法，可以用于将数据存储到指定的文件中。

所有的文件会默认存储到/data/data/<package name>/files/目录下。写法如下：

```
fun save(inputText: String) {
try {
val output = openFileOutput("data", Context.MODE_PRIVATE)
val writer = BufferedWriter(OutputStreamWriter(output))
writer.use {
it.write(inputText)
}
} catch (e: IOException) {
e.printStackTrace()
}
```

```
        }
```

② 从文件中读取数据：

Context 类中还提供了一个 openFileInput()方法，用于从文件中读取数据。

它会自动到/data/data/<package name>/files/目录下加载文件，并返回一个 FileInputStream 对象，得到这个对象之后，再通过流的方式就可以将数据读取出来了。写法如下：

```
fun load(): String {
    val content = StringBuilder()
    try {
        val input = openFileInput("data")
        val reader = BufferedReader(InputStreamReader(input))
        reader.use {
            reader.forEachLine {
                content.append(it)
            }
        }
    } catch (e: IOException) {
        e.printStackTrace()
    }
    return content.toString()
}
```

3. SharedPreferences 存储

不同于文件的存储方式，SharedPreferences 是使用键值对的方式来存储数据的。

也就是说，当保存一条数据的时候，需要给这条数据提供一个对应的键，这样在读取数据的时候就可以通过这个键把相应的值取出来。

SharedPreferences 还支持多种不同的数据类型存储，如果存储的数据类型是整型，那么读取出来的数据也是整型的；如果存储的数据是一个字符串，那么读取出来的数据仍然是字符串。

① 将数据存储到 SharedPreferences 中

向 SharedPreferences 文件中存储数据了，主要可以分为 3 步实现：

1. 调用 SharedPreferences 对象的 edit()方法获取一个 SharedPreferences.Editor 对象。
2. 向 SharedPreferences.Editor 对象中添加数据，比如添加一个布尔型数据就使用 putBoolean()方法，添加一个字符串则使用 putString()方法，以此类推。
3. 调用 apply()方法将添加的数据提交，从而完成数据存储操作。

示例写法如下：

```
val editor = getSharedPreferences("data", Context.MODE_PRIVATE).edit()
editor.putString("name", "Tom")
```

```
            editor.putInt("age", 28)
            editor.putBoolean("married", false)
            editor.apply()
```

② 从 SharedPreferences 中读取数据

SharedPreferences 对象中提供了一系列的 get 方法，用于对存储的数据进行读取，每种 get 方法都对应了 SharedPreferences.Editor 中的一种 put 方法。

比如读取一个布尔型数据就使用 getBoolean()方法，读取一个字符串就使用 getString()方法。

示例写法如下：

```
val prefs = getSharedPreferences("data", Context.MODE_PRIVATE)
val name = prefs.getString("name", "")
val age = prefs.getInt("age", 0)
val married = prefs.getBoolean("married", false)
```

4. 数据库存储

① 创建数据库

Android 为了让我们能够更加方便地管理数据库，专门提供了一个 SQLiteOpenHelper 帮助类，借助这个类可以非常简单地对数据库进行创建和升级。示例写法如下：

```
class MyDatabaseHelper(val context: Context, name: String, version: Int) :
SQLiteOpenHelper(context, name, null, version) {
    private val createBook = "create table Book (" +
            "id integer primary key autoincrement," +
            "author text," +
            "price real," +
            "pages integer," +
            "name text)"
    override fun onCreate(db: SQLiteDatabase) {
        db.execSQL(createBook)
        Toast.makeText(context, "Create succeeded", Toast.LENGTH_SHORT).show()
    }
    override fun onUpgrade(db: SQLiteDatabase, oldVersion: Int, newVersion: Int) {
    }
}
```

② 升级数据库

onUpgrade()方法是用于对数据库进行升级的，它在整个数据库的管理工作当中起着非常重要的作用。示例写法如下：

```
class MyDatabaseHelper(val context: Context, name: String, version: Int) :
```

```
SQLiteOpenHelper(context, name, null, version) {
    private val createCategory = "create table Category (" +
            "id integer primary key autoincrement," +
            "category_name text," +
            "category_code integer)"
    override fun onCreate(db: SQLiteDatabase) {
        db.execSQL(createBook)
        db.execSQL(createCategory)
        Toast.makeText(context, "Create succeeded", Toast.LENGTH_SHORT).show()
    }
    override fun onUpgrade(db: SQLiteDatabase, oldVersion: Int, newVersion: Int) {
        db.execSQL("drop table if exists Book")
        db.execSQL("drop table if exists Category")
        onCreate(db)
    }
}
```

③ 添加数据

第一个参数是表名，我们希望向哪张表里添加数据，这里就传入该表的名字；第二个参数用于在未指定添加数据的情况下给某些可为空的列自动赋值 NULL，一般我们用不到这个功能，直接传入 null 即可；第三个参数是一个 ContentValues 对象，它提供了一系列的 put()方法重载，用于向 ContentValues 中添加数据，只需要将表中的每个列名以及相应的待添加数据传入即可。

示例写法如下：

```
val dbHelper = MyDatabaseHelper(this, "BookStore.db", 2)
val db = dbHelper.writableDatabase
val values1 = ContentValues().apply {
    // 开始组装第一条数据
    put("name", "The Da Vinci Code")
    put("author", "Dan Brown")
    put("pages", 454)
    put("price", 16.96)
}
db.insert("Book", null, values1) // 插入一条数据
```

④ 更新数据

```
val dbHelper = MyDatabaseHelper(this, "BookStore.db", 2)
val db = dbHelper.writableDatabase
val values = ContentValues()
values.put("price", 10.99)
db.update("Book", values, "name = ?", arrayOf("The Da Vinci Code"))
```

⑤ 删除数据

```
val dbHelper = MyDatabaseHelper(this, "BookStore.db", 2)
```

```
val db = dbHelper.writableDatabase
db.delete("Book", "pages > ?", arrayOf("500"))
```

⑥ 查询数据

```
val dbHelper = MyDatabaseHelper(this, "BookStore.db", 2)
val db = dbHelper.writableDatabase
// 查询 Book 表中所有的数据
val cursor = db.query("Book", null, null, null, null, null, null)
if (cursor.moveToFirst()) {
    do {
        // 遍历 Cursor 对象，取出数据并打印
        val name = cursor.getString(cursor.getColumnIndex("name"))
        val author = cursor.getString(cursor.getColumnIndex("author"))
        val pages = cursor.getInt(cursor.getColumnIndex("pages"))
        val price = cursor.getDouble(cursor.getColumnIndex("price"))
        Log.d("MainActivity", "book name is $name")
        Log.d("MainActivity", "book author is $author")
        Log.d("MainActivity", "book pages is $pages")
        Log.d("MainActivity", "book price is $price")
    } while (cursor.moveToNext())
}
cursor.close()
```

⑦ 使用 SQL 操作数据库

1. 添加数据：

```
db.execSQL(
    “insert into Book (name, author, pages, price) values(?, ?, ?, ?)”,
arrayOf("The Da Vinci Code", "Dan Brown", "454", "16.96")
)
```

2. 更新数据：

```
db.execSQL("update Book set price = ? where name = ?", arrayOf("10.99", "The Da Vinci Code"))
```

3. 删除数据：

```
db.execSQL("delete from Book where pages > ?", arrayOf("500"))
```

4. 查询数据：

```
val cursor = db.rawQuery("select * from Book", null)
```

【教师评语和成绩】

成绩： **指导教师：** **日期：**